1990

Supporting Queries in the O-Raid Object-Oriented Database System

James G. Mullen

Jagannathan Srinivasan

Prasun Dewan

Bharat Bhargava
Purdue University, bb@cs.purdue.edu

Report Number:
90-956

SUPPORTING QUERIES IN THE O-RAID OBJECT-ORIENTED DATABASE SYSTEM

James G. Mullen
Jagannathan Srinivasan
Prasun Dewan
Bharat Bhargava

CSD TR-956
February 1990

Supporting Queries in the O-Raid Object-Oriented Database System *

James G. Mullen Jagannathan Srinivasan
Prasun Dewan Bharat Bhargava

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

Abstract

We present the query language SQL++ (an extension of the relational query language SQL), and discuss its capabilities in the O-Raid object-oriented database system. We show the advantages of the object-relation model used in O-Raid for supporting SQL++. In O-Raid objects are organized into relations. The advantages of this organization for supporting ad-hoc queries include: ease of grouping data by its logical view, ease of handling and storing the results of query operations, and upward compatibility with relational database systems. The advantages over the relational systems for supporting queries include: use of object pointers to perform implicit joins, support for navigational queries, and support for implementing triggers in the database by using object methods.

*This research is supported in part by NASA, AIRMICS, and the UNISYS Corporation.

1 Introduction

Object-oriented database systems have been developed to support applications that require a more complex data model, such as CAD/CAM and AI applications [SR87, BCG+87], and to provide support for shared persistent objects in object-oriented programming languages [DVB89, KBC+88].

Two of the complaints of the object-oriented databases is that they return us to the days of "navigational databases" where one must specify *how to get* the information desired as opposed to specifying *what* information is desired, and that existing applications have to be re-written as an entirely new approach is taken with a different data model [Ull87, BZ87].

O-Raid [DVB89, BDMS90] is the object-oriented version of the RAID distributed database system [BR89], and avoids the two problems mentioned above by using the object-relation model [Rum87] where data are organized into relations of objects, which can be arbitrarily complex. More specifically, the O-Raid data model extends the relational model by allowing attributes in a relation to belong to user defined types, where the user defined types can be as complex as in a general purpose object-oriented programming language. This model eases the task of supporting ad-hoc queries in an object-oriented system, and enables O-Raid to be upwardly compatible with relational database systems, since the data model used is an extension (or generalization) of the relational data model.

O-Raid provides benefits for query support as in other object-oriented database systems. The support of object pointers in O-Raid allows implicit joins to be performed, and provides support for navigational queries, which is useful, for instance, for hypertext applications [SZ87, Con87]. Finally, methods of objects can be used as triggers.

Section 2 gives an overview of the O-Raid system. Section 3 discusses how our data model can provide better support for queries than other object-oriented data models, and section 4 discusses how our data model can provide better support for queries than the purely relational model. Section 5 points out how O-Raid differs from other object-oriented database systems, and other systems which extend the relational data model. Section 6 presents ideas for future work and conclusions.

2 Overview of O-Raid

Data Model. O-Raid [DVB89, BDMS90] is an object-oriented database system that uses the object-relation model. Data items are organized into relations as in the relational case. However, unlike the relational case where the type of a data item is restricted to a small set of simple pre-defined types, the data items in O-Raid can be arbitrarily complex objects (as in a general purpose object-oriented programming language)[1]. O-Raid still supports the same simple pre-defined types as the relational data model, so the object-relation model supported by O-Raid is a generalization of the relational data

[1] In fact, O-Raid uses C++ as its Object Definition Language.

model.

SQL++ SQL is being perceived as a standard query language for relational database systems [Dat86]. SQL++ is the query language used in O-Raid and is an extension of SQL to support complex data types of the object model for databases.

SQL++ data definition facilities. SQL++ supports data definition facilities as in SQL. For example to create a new relation the following could be entered:

```
CREATE TABLE employees (char[24] name, int id, int dept_id);
```

In SQL++ the types of the attributes are not restricted to being primitive types as in the example above, but could also be user-defined types (or classes), as shown in the following command:

```
CREATE TABLE employees (char[24] name, int id, Department *dept);
```

In this example the attribute *dept* is declared to be a pointer to type *Department*.

In O-Raid, user-defined types (classes) are defined using the C++ programming language [Str86]. The definition of the new type is written in C++, and then the new type is registered with the database using the *class-create* utility. For example, the *Department* class used above could be defined as follows:

```
class Department {
public:
    char name[20];
    Department(char *name);
};
```

"name[20]" is a character array used to hold the name of the department. "Department(char *name)" is a *constructor method* or simply a *constructor*. It is used to create an object of type *Department*. The definition of the constructor is typically placed in a separate file, and is not shown here.

SQL++ data manipulation facilities. SQL++ supports data manipulation commands as in SQL, such as INSERT, DELETE, and UPDATE. In SQL when a new tuple is inserted into a relation the new values are simply listed in the command. This syntax needs to be extended in our system to support complex objects. For the user-defined objects, the *constructor method* is used to initialize the object. In C++, objects have a method (or function) called a constructor, which has the same name as the class of the object. An example of inserting a tuple into the relation *employees* where one of the attributes is a pointer to type *Department* would be as follows:

```
INSERT INTO employees :
    < "John Johnson", 159, &Department("Marketing") >
```

The "&" is used before the constructor to indicate that the address of the Department object should be returned, instead of the object itself.

SQL++ query facilities. The query language commands used in SQL++ are the same as in SQL but contain extensions to support queries involving objects. Since objects are defined using C++, we have adopted the C++ syntax in the query language to reference subobjects and methods of objects. For a more complete description of the SQL syntax see [CAE+76], and for details of the C++ syntax see [Str86]. As an example, suppose a user wants to print the name of the department of employee "John Johnson". The following query is used, assuming the above definitions:

```
SELECT employees.dept->name FROM employees
    WHERE employees.name = "John Johnson";
```

The following query is used to retrieve the information on all employees who work in the Research group:

```
SELECT * FROM employees
    WHERE employees.dept->name = "Research";
```

In addition to supporting relational queries, O-Raid supports object pointers which allow navigational queries to be performed. Assume that one wants to store a list of names, and that a relation *first* has been created that holds the first name in the list, and a relation *names* exists that holds the other names in the list. The attributes of these relations are *name*, and *next* which is a pointer to the next name in the list. The following would be an example of a navigational query:

```
ASSIGN TO temp: SELECT * FROM first;
```

```
SELECT name FROM temp;
ASSIGN TO temp: SELECT *next FROM temp;
SELECT name FROM temp;
ASSIGN TO temp: SELECT *next FROM temp;
SELECT name FROM temp;
```

The above query prints out the first three names on the list of names. The first assignment statement gets the first name in the list, the second assignment statement follows the pointer to the next name in *temp* and resets *temp* to the value pointed to. The "SELECT name FROM temp;" statements print out the name.

It is possible to invoke methods of objects in a SELECT statement, for example[2]:

```
SELECT picture.display() FROM students;
```

If one assumes that the methods invoked will not have any side-effects and will eventually terminate there will be no problems. However, if one cannot make this assumption then it is possible that a query could never terminate, or could get into a deadlock.

O-Raid interface to Application Programs. We provide a preprocessor for C++ programs so that SQL++ queries can be embedded in the programs. Since the objects in our database are defined using C++, they are compatible

[2]This example assumes that a relation *students* exists that has an attribute named *picture* that has a method named *display*

with C++ objects. This eliminates the need of an application program to translate from one data format to another when retrieving information from, or storing information to the database (the "impedance mismatch" problem). An example C++ program fragment is given below:

```
## Department department("");

## SELECT $department = dept FROM employees
    ## WHERE employees.name = "John Johnson";
```

In this example a variable called *department* is declared, which has type *Department*. A query is made that retrieves John Johnson's department and assigns it to the variable *department*. We prepend SQL++ statements and C++ variable declarations where the variable is to be used in an SQL++ statement by "##" as is done in EQUEL [Sto86]. The "*" is used before *dept* to indicate that the value of *dept* should be returned instead of its address. This is necessary since the attribute *employees.dept* is an object pointer.

Implementation of SQL++ interface We have implemented the parser for a subset of SQL++, by extending the SQL grammar to allow subobject and method references to be made using C++ syntax. Subobjects and methods are referenced using dot notation if they are contained directly within an object, or arrows ("->") if the subobject or method is pointed to. The size of the source code of the O-Raid SQL++ parser about 18K bytes.

3 Advantages of Organizing Objects into Relations

Logical Grouping of Objects. O-Raid allows a user to group different types of objects in a single relation. Thus a user can define relations to organize objects in a manner that are easy to understand. The O-Raid approach has potential performance benefits as relations allow clustering of related data. In contrast storing objects of a class together, as is done in some object-oriented systems, may lead to a large collection of objects making it inefficient to perform queries dealing with a small subset of that collection.

Uniform Data Structure. The object-relation model supports a uniform data structure (the relation) used to group objects. This facilitates the implementation of standard operations on groups of data. In O-Raid these operations are from relational algebra, that is: projection, selection, cartesian product (or join), difference, and union, etc... The operands and result of these operations are always relations. In other systems the type of operands and results may vary, and the type of the result may be an undefined type. Object-oriented systems that support these operations must implement them for all possible classes, and must have a way to dynamically change the class hierarchy when the result of a query has a previously undefined type.

Although relations can be simulated in systems that store objects by class and have no relations, by declaring a new class that has as members the information that one wants grouped together, a problem arises when some

certain queries are performed. For example, when a projection is performed that eliminates some of the members of a class, the result will have a structure that may not be defined as a class. Even if it was predefined, it might be costly to determine. One approach is to dynamically modify the class hierarchy to add the class of the result. However, in a multi-user database environment this will make the class hierarchy data structure a hot spot item, since even simple queries might need to update the class hierarchy. In addition, there are some other problems related to the choice of positioning of the new class in the class hierarchy that are discussed in [Kim89]. Since the projection creates new instances of a class, and many object-oriented databases require all objects to have a unique object identifier, an object identifier must be requested for *each* new instance. In addition, for most systems, a new entry will have to be added to an address translation table that translates object identifiers to physical addresses. Perhaps these problems can be avoided if the results of an operation are only temporary (either used to calculate a final result, or only displayed to a terminal) by taking some short-cuts. However, when the results of a query are stored, for example[3]:

ASSIGN TO names: SELECT employees.name FROM employees

these problems cannot be avoided. The same problems for projections occur when performing a join, because one may need to create a new class for the result, and there will be some overhead for setting up the instances of the new class.

[3]This command stores all the names from the employees relation in a new relation called *names*

Support for Relational Queries. O-Raid can support "standard" relational queries because the underlying data model is a generalization of the relational data model and the query language is a superset of SQL. Thus, unlike most other object-oriented database systems, O-Raid can be used to manage purely relational databases and can support existing relational applications without any changes. The data definition, data manipulation, and query facilities of SQL++ are all supersets of those in SQL.

4 Comparison with Relational Systems

Implicit joins. In addition to being able to support standard relational applications with our system, we are also able to support them in improved ways. We support pointers to objects, which supports the idea of an *implicit join.* Such a join can save storage, and help reduce update anomalies, like normalization in a relational system, but avoids requiring the user to explicitly join relations when performing queries.

An example is given where a relation of employees and a relation of departments exists. In the relational model department id's must be used and the two relations joined together to get the name of the department listed for the employee. In the object-relation case where pointers are used, the department name is automatically printed with the employee name, without the need to perform a join. In addition, in the relational case, attributes may need to be created that do not correspond to any real world information, but are necessary to perform the join. For example, if the organization

has no concept of a department id, and departments are always referenced by a name or an abbreviation, then the department id attribute would be such an attribute. To illustrate, let us consider the following query in a relational system:

```
SELECT * FROM employees, dept
    WHERE dept.id = employees.dept_id AND
            employees.name = "John Johnson";
```

and the following in an equivalent query in O-Raid performing an implicit join:

```
SELECT * FROM employees
    WHERE employees.name = "John Johnson";
```

Figure 1 shows how the data for the relational case would be organized, and figure 2 shows how the data would be organized in O-Raid to use an implicit join.

Navigational queries. Object pointers allow navigational queries to be performed in O-Raid. These types of queries have been shown to be useful, for instance, for supporting hypertext applications [SZ87, Con87]. They can be performed in SQL++ by iteratively assigning the result of a query to a relation, and then performing a query on the result to get a new result. Although this scheme will work, we plan on providing a more user-friendly graphical interface for performing such searches [Dewue].

EMPLOYEE RELATION

name	id	dept_id
Joe Smith	123	101
Sue Smith	144	101
John Smith	124	102
John Johnson	159	101
Jacky Jackson	187	102
Jill Smith	118	102

DEPARTMENT RELATION

id	name
101	Marketing
102	Research

Figure 1: Relational Data Model.

EMPLOYEE RELATION

name	id	department
Joe Smith	123	
Sue Smith	144	
John Smith	124	
John Johnson	159	
Jacky Jackson	187	
Jill Smith	118	

DEPARTMENT RELATION

name
Marketing
Research

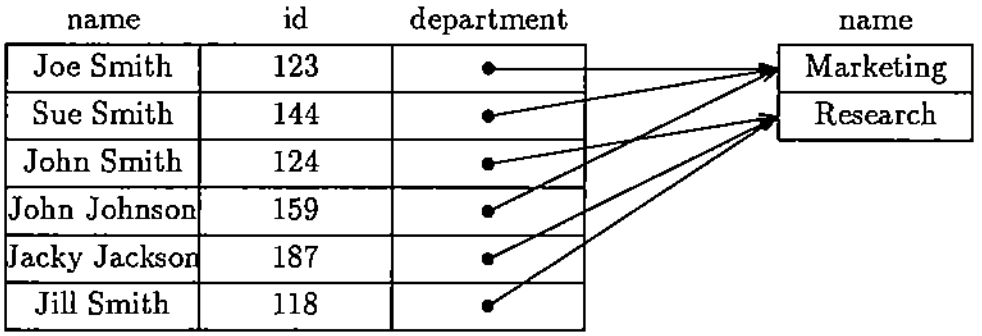

Figure 2: O-Raid Data Model Using Object Pointers.

Support for triggers. Object methods can be used as triggers in the database. For example, a delete_dept method could be written for the Department object in the previous example that would delete all the employees that belonged to a department when the department is deleted.

5 Related Work

The object-relation model has been discussed and implemented in an object-oriented programming language system [Rum87]. The motivation being that certain relationships can be expressed more clearly by using relations then through using object pointers. The language has built-in object classes to support relations. The methods supported for relation class include operations for adding, deleting, scanning, and indexing the relation tuples. The experiences in [Rum87] in building applications using this approach suggests that use of relations greatly enhance the modeling process.

There are other systems which have tried to extend the relational system to enhance its power, such as RAD [OH86] and ADT-INGRES [Sto86], which extend the allowable types of attributes in relations. However, these systems are not object-oriented, since they do not support inheritance or encapsulation of methods within objects. POSTGRES [SR87, RS87] extends the relational model by extending the types of attributes, and does support inheritance. In POSTGRES attributes may have a primitive type (integer, floating point, or fixed length character string), type POSTQUEL or procedure, and can be arrays of fixed types with an arbitrary number of di-

mensions. Type POSTQUEL is a sequence of data manipulation commands, type procedure is a procedure written in a general purpose programming language. In O-Raid all columns in relations contain objects. The objects may contain variables and methods (or procedures) written in a general purpose programming language, and the methods may contain embedded query language statements. So, POSTGRES extends relations differently than we do, and does not support the encapsulation of methods within objects.

Our approach of extending an existing relational query language to handle object-oriented queries is also used by other systems. The Iris database system [FAB+88] also uses a query language that is an adaptation of SQL called OSQL (Object SQL), however it operates on types (classes) and functions, and not on relations, so it does not use the object relation model, and suffers from some of the problems listed earlier in the paper. In addition, the data definition facilities of SQL are not supported. EXODUS [CD88] uses a query language that is an adaptation to QUEL. Although relations are not implemented, sets, and fixed and variable sized arrays are implemented, so relation like constructs can be created. While EXODUS solves the problem of grouping data by its logical view instead of by class, and seems to be relatively upwardly compatible with a relational database, it still has problems storing query results as we discussed. Also, since there is no concept of relations, data definition must be done in a different way.

6 Future Work and Conclusions

Future Work. We plan to run experiments to get performance data on our system for processing relational queries, and for processing queries that use complex objects. We plan to use the Wisconsin benchmarks [BDT83] to test relational query processing, and to test object-oriented query processing by extending these benchmarks by using complex user-defined types as attributes in the relations. We will vary the number and complexity of the user-defined types used, and examine the effect on performance for CPU processing time, and I/O time. We plan to implement query optimization methods and query processing methods for object-oriented queries, and to use the experiments mentioned above to test their effectiveness. Also, we will extend the data definition facilities to allow for partial replication of data.

Conclusions. The use of the object-relation model in the O-Raid system allows it to model complex objects while still being upwardly compatible with standard relational systems. It also eases the task of supporting ad-hoc object-oriented queries, and retains the advantages of object-oriented systems over relational systems for supporting queries.

References

[BCG+87] Jay Banerjee, Hong-Tai Chou, Jorge F. Garza, Won Kim, Darrel Woelk, Nat Ballou, and Hyoung-Joo Kim. Data Model Issues for Object-Oriented Applications. *ACM Transactions on Office Information Systems*, 5, Jan 1987.

[BDMS90] Bharat Bhargava, Prasun Dewan, James G. Mullen, and Jagannathan Srinivasan. Implementing Object Support in the RAID Distributed Database System. In *Proceedings Of The First International Conference on Systems Integration*, 1990. to appear.

[BDT83] D. Bitton, D.J. DeWitt, and C. Turbyfil. Benchmarking Database Systems: a Systematic Approach. In *Proceedings of the VLDB Conference*, October 1983.

[BR89] Bharat Bhargava and John Riedl. The RAID Distributed Database System. *IEEE Transactions on Software Engineering*, 15(6), June 1989.

[BZ87] Toby Bloom and Stanley B. Zdonik. Issues in the Design of Object-Oriented Database Programming Languages. In *OOPSLA 1987 Proceedings*, pages 441–451, October 1987.

[CAE+76] D. D. Chamberlin, M. M. Astrahan, K. P. Eswaran, P. P. Griffiths, R. A. Lorie, J. W. Mehl, P. Reisner, and B. W. Wade. SEQUEL 2: A Unified Approach to Data Definition, Manipulation, and Control. *IBM Journal of Research and Development*, pages 560–575, November 1976.

[CD88] Michael J. Carey and David J. DeWitt. A Data Model and Query Language for EXODUS. In *Proceedings of the SIGMOD International Conference on Management of Data*, Jun 1988.

[Con87] Jeff Conklin. Hypertext: An Introduction and Survey. *IEEE Computer*, 20(9), September 1987.

[Dat86] C. J. Date. *An Introduction to Database Systems*, volume 1. Addison-Wesley, 1986.

[Dewue] Prasun Dewan. Object-Oriented Editor Generation. *Journal of Object-Oriented Programming*, expected to appear in the May/June 1990 issue.

[DVB89] Prasun Dewan, Ashish Vikram, and Bharat Bhargava. Engineering the Object-Relation Model in O-Raid. In *Proceedings Of The*

International Conference on Foundations of Data Organization and Algorithms, pages 389–403, June 1989.

[FAB+88] Daniel H. Fishman, Jurgen Annevelink, David Beech, et al. Overview of the Iris DBMS. Technical report, Database Technology Department, Hewlett-Packard Laboratories, Jun 1988.

[KBC+88] Won Kim, Nat Ballou, Hong-Tai Chou, Jorge F. Garza, Darrel Woelk, and Jay Banerjee. Integrating an Object-Oriented Programming System with a Database System. In *Proceedings of OOPSLA '88*, Sep 1988.

[Kim89] Won Kim. A Model of Queries for Object-Oriented Databases. In *Proceedings of the Fifteenth International Conference on Very Large Data Bases*, pages 423–432, 1989.

[OH86] Sylvia L. Osborn and T. E. Heaven. The Design of a Relational Database System with Abstract Data Types for Domains. *ACM Transactions on Database Systems*, 11(3):357–373, September 1986.

[RS87] Lawrence A. Rowe and Michael Stonebraker. The POSTGRES Data Model. In *Proc. 13th VLDB Conference*, 1987.

[Rum87] James Rumbaugh. Relations as Semantic Constructs in an Object-Oriented Language. In *Proceedings of OOPSLA '87*, pages 466–481, October 1987.

[SR87] Michael Stonebraker and Lawrence A. Rowe. The POSTGRES Papers. Technical Report UCB/ERL M86/85, University of California, Berkeley, Jun 1987.

[Sto86] Michael Stonebraker, editor. *The INGRES Papers: Anatomy of a Relational Database System.* Addison-Wesley, 1986.

[Str86] Bjarne Stroustrup. *The C++ Programming Language.* Addison-Wesley, Reading, Mass., 1986.

[SZ87] Karen E. Smith and Stanley B. Zdonik. Intermedia: A Case Study of the Differences Between Relational and Object-Oriented

Database Systems. In *OOPSLA '87 Proceedings*, pages 452–465, October 1987.

[Ull87] J. Ullman. Database Theory - Past and Future. In *Proceedings of the PODS Conference, San Diego, CA.*, March 1987.